**Отчет о лабораторной работе 3 по курсу «Разработка ПО ОС UNIX”**

**Тема: «Удаленный вызов процедур»**

Выполнил:

Студент группы Б13-001

Спирихин М.А.

1. **Задание**

Разработать программу для Solaris. Программа состоит из серверной и клиентской частей. Серверная часть делит полином на полином, клиентская часть вводит исходные данные и выводит результат. Программа должна собираться из нескольких файлов с использованием make.

1. **О программе**

Программа представляет собой утилиту для командной строки и выполняет некоторое действие в зависимости от аргументов

-с key – создать память по ключу

-f key – проверить существование памяти

-r key – прочитать из памяти

-w key value – записать в память

-d key – удалить память

Без аргументов или некорректные аргументы – показать справку

Внутри программы происходит работа с разделяемой памятью System V. В разделяемую память помещается строка из 255 символов. Для обеспечения одновременного доступа к памяти нескольких экземпляров программы используется семафор.

1. **Исходный код**

#define SHAR\_MEM\_SIZE 255\*sizeof(char)

#define SEMAPHORE\_KEY 12321

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <stdio.h>

#include <errno.h>

#include <sys/sem.h>

#include <unistd.h>

#include <string.h>

//Create of check shared memory

**int** **CreateOrCheck**(**key\_t** key)

{

**int** shmid;

**if**((shmid = shmget(key, SHAR\_MEM\_SIZE, **0666**|IPC\_CREAT|IPC\_EXCL)) >= **0**){

printf("Shared memory with key %d created successfully**\n**",key);

} **else** {

**if**(errno == EEXIST){

**if**((shmid = shmget(key, SHAR\_MEM\_SIZE, **0**)) >= **0**){

printf("Shared memory with key %d is already exists**\n**",key);

} **else** {

printf("Shared memory with key %d has a wrong size**\n**",key);

**return** -**1**;

}

} **else** {

printf("Can**\'**t create shared memory (errno = %d)**\n**",errno);

**return** -**1**;

}

}

**return** **0**;

}

//check shared memory and return shared memory id

**int** **FindSharedMemory**(**key\_t** key)

{

**int** shmid;

**if**((shmid = shmget(key, SHAR\_MEM\_SIZE, **0**)) >= **0**){

**return** shmid;

} **else** {

**return** -**1**;

}

}

//read the value from shared memory and return it

**char**\* **ReadFromSharedMemory**(**int** shmid)

{

**char**\* str;

**if**((str = (**char** \*)shmat(shmid, NULL, **0**)) == (**char** \*)(-**1**)){

printf("Can't attach shared memory**\n**");

**return** (**char** \*)-**1**;

}

**return** str;

}

//write value to shared memory and return 0 if success, -1 if error

**int** **WriteToSharedMemory**(**int** shmid, **char**\* value)

{

**char**\* str;

**if**((str = (**char** \*)shmat(shmid, NULL, **0**)) == (**char** \*)(-**1**)){

printf("Can't attach shared memory**\n**");

**return** -**1**;

}

strcpy(str,value);

**if**(shmdt(str) < **0**){

printf("Can't detach shared memory**\n**");

**return** -**1**;

}

**return** **0**;

}

//delete shared memory and return 0 if success, -1 if error

**int** **DeleteSharedMemory**(**int** shmid)

{

**if**(shmctl (shmid, IPC\_RMID, NULL) < **0**){

printf("Error while deleting shared memory**\n**");

**return** -**1**;

}

**return** **0**;

}

//do Action of program (0 - OK, 1 - wrong input)

**int** **doAction**(**char** arg, **int** key, **char**\* str)

{

**if** (arg == 'c') {

CreateOrCheck(key);

**return** **0**;

} **else** **if** (arg == 'r') {

**int** shmid = FindSharedMemory(key);

**if** (shmid < **0**)

{

printf("Shared memory with key %d is not exists**\n**",key);

**return** **0**;

}

**char**\* val = ReadFromSharedMemory(shmid);

printf("Value=**\"**%s**\"\n**",val);

**return** **0**;

} **else** **if** (arg == 'f') {

**int** shmid = FindSharedMemory(key);

**if** (shmid >= **0**)

printf("Shared memory with key %d is exists**\n**",key);

**else**

printf("Shared memory with key %d is not exists**\n**",key);

**return** **0**;

} **else** **if** (arg == 'w') {

**if** (str == (**char**\*)NULL)

**return** **1**;

**int** shmid = FindSharedMemory(key);

**if** (shmid < **0**)

{

printf("Shared memory with key %d is not exists**\n**",key);

**return** **0**;

}

**if** (WriteToSharedMemory(shmid,str) == **0**);

printf("**\"**%s**\"** has written**\n**",str);

**return** **0**;

} **else** **if** (arg == 'd') {

**int** shmid = FindSharedMemory(key);

**if** (shmid < **0**)

{

printf("Shared memory with key %d is not exists**\n**",key);

**return** **0**;

}

**if** (DeleteSharedMemory(shmid) == **0**)

printf("Shared mamory with key **\"**%d**\"** was successfully deleted**\n**",key);

**return** **0**;

}

**return** **1**;

}

**int** **PrintHelp**(**char**\* name)

{

printf("%s usage:**\n**", name);

printf("%s -c key - create or check existanse of shared memory with key**\n**", name);

printf("%s -f key - check existanse of shared memory with key**\n**", name);

printf("%s -r key - read shared memory value**\n**", name);

printf("%s -w key value - write value to shared memory**\n**", name);

printf("%s -d key - delete shared memory with key**\n\n**", name);

**return** **0**;

}

**int** **main**(**int** argc, **char**\*\* argv)

{

//check correct

**if** (argc < **3**)

**return** PrintHelp(argv[**0**]);

**if** (strlen(argv[**1**])!=**2** || argv[**1**][**0**] != '-')

**return** PrintHelp(argv[**0**]);

//read key

errno = **0**;

**char**\* p;

**int** key = strtol(argv[**2**], &p, **10**);

**if** (errno != **0** || \*p != '\0') {

printf("Can**\'**t turn **\"**%s**\"** into int",argv[**2**]);

**return** **0**;

}

//read value

**char**\* value = NULL;

**if** (argc > **3**)

value = argv[**3**];

//------Activate semaphore and do action------

//create or get semaphore

**int** semid;

**if**((semid = semget(SEMAPHORE\_KEY, **1**, **0666** | IPC\_CREAT)) < **0**){

printf("Can**\'**t get semaphore**\n**");

**return** -**1**;

}

**struct** sembuf mybuf;

mybuf.sem\_op = **0**;

mybuf.sem\_flg = **0**;

mybuf.sem\_num = **0**;

semop(semid, &mybuf, **1**);//wait semaphore

mybuf.sem\_op = **1**;

semop(semid, &mybuf, **1**);//block semaphore

**int** res = doAction(argv[**1**][**1**],key,value);

//sleep(4);

mybuf.sem\_op = -**1**;

semop(semid, &mybuf, **1**);//unblock semaphore

**if** (res == **1**)

**return** PrintHelp(argv[**0**]);

}